Detecting Road Intersections from GPS Traces

Alireza Fathi[1] and John Krumm[2]

[1] College of Computing
Georgia Institute of Technology
Atlanta, Georgia USA
alireza.fathi@gmail.com
[2] Microsoft Research
Microsoft Corporation
Redmond, Washington USA
jckrumm@microsoft.com

Abstract. As an alternative to expensive road surveys, we are working toward a method to infer the road network from GPS data logged from regular vehicles. One of the most important components of this problem is to find road intersections. We introduce an intersection detector that uses a localized shape descriptor to represent the distribution of GPS traces around a point. A classifier is trained on the shape descriptor to discriminate intersections from non-intersections, and we demonstrate its effectiveness with an ROC curve. In a second step, we use the GPS data to prune the detected intersections and connect them with geometrically accurate road segments. In the final step, we use the iterative closest point algorithm to more accurately localize the position of each intersection. We train and test our method on GPS data gathered from regular vehicles in the Seattle, WA, USA area. The tests show we can correctly find road intersections.

Keywords: GPS, road map, road network, intersection detection.

1 Introduction

Digital road maps are clearly important for both consumers and businesses. At present, these maps are created by companies fielding fleets of specialized vehicles equipped with GPS to drive the roads and record data. This is an expensive process, and it is difficult to keep up with changes in the road network. An emerging alternative is to use GPS data from regular vehicles driving their regular routes. This has the advantage of easily scaling to the entire road network and providing much more up-to-date data whenever roads change.

The challenge of this technique is how to process all the data into a road map. The OpenStreetMap [1] project provides one model where volunteers manually edit GPS traces and aerial images into digital maps. While OpenStreetMap moves away from the use of specialized vehicles, we would like to eliminate the manual step. In this paper, we show how to automate one important aspect of this processing: finding road intersections. We test our process on a large amount of GPS data we gathered from

S.I. Fabrikant et al. (Eds.): GIScience 2010, LNCS 6292, pp. 56–69, 2010.

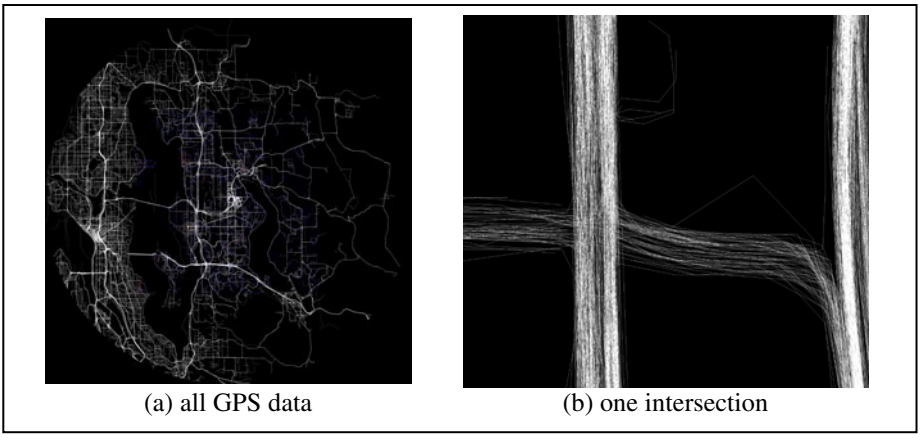


Fig. 1. GPS traces. (a) shows an overview of our GPS traces from the greater Seattle, WA USA area. In (b) is a close-up of traces around a road intersection.

vehicles that were already driving in our metropolitan area. This data is shown in Fig. 1(a).

Our algorithm detects intersections in the GPS data, an example of which is shown in Fig. 1(b). It begins by using a shape descriptor trained on positive and negative examples of intersections. Next it connects the intersections by finding vehicle traces that move between them. Finally, the algorithm refines the locations of the intersections based on the GPS data associated with the nearby roads. We evaluate our algorithm by comparing it to a known road network. Specifically, we evaluate it in terms of its ability to find intersections, the accuracy of the intersections' computed locations, and the accuracy of the lengths of the roads between the intersections.

2 Previous Work

Some of the earliest research into building road maps was based on aerial images, such as that by Tavakoli & Rosenfeld [2]. They group together edges found by an edge detector into shapes representing buildings and roads. Hu *et al.* [3] find seed pixels that were likely roads and then grew from the seeds by tracking along road segments. Barsi & Heipke [4] trained a neural network to find intersections in images. This is related to our work in that we also train an intersection detector, although ours is formulated differently and works on GPS traces rather than image pixels.

Another line of research addresses the problem of refining an existing map using GPS traces. Rogers *et al.* [5] use an initial map to refine the centerline of the road. In this work, they look at perpendiculars to the refined centerline and cluster traces into lanes. Guo *et al.* [6] present initial simulation work with a similar goal of finding the centerline of the road. Our goal is to build a road map without any prior road map.

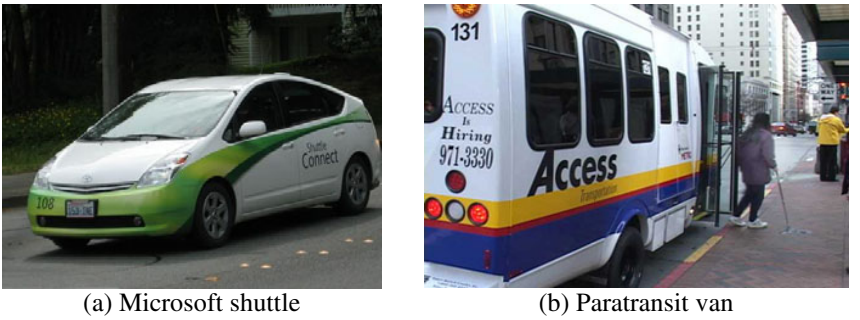
There are other efforts with the same goal. Two of these, Brüntrup *et al.* [7] and Worral & Nebot [8] present simple clustering techniques for determining the location of the road. Some of the deepest work on this problem comes from Schroedl *et al.* [9]

and Edelkamp & Schrodel [10]. Starting with a blank map, they first find centerlines by clustering. Then they determine the structure of the lanes and intersections. Recent work by Cao and Krumm [11] builds a routable road network by first clarifying the GPS traces and then clustering them into a connected graph representing the roads.

Our approach is different than the above in that we begin by finding intersections using a detector trained on ground truth data. As far as we are aware, this represents the first use of a trained detector to find road features from GPS data. After finding intersections, we discover connecting edges by looking for contiguous traces going between them. We use these connecting edges to refine the location of the detected intersections. Before we discuss our technical approach, however, we present in the following section the GPS data we used for our training and testing.

3 GPS Data

We have collected GPS data for testing by deploying GPS loggers on 55 Microsoft Shuttles and 252 King County Paratransit vehicles, as depicted in Fig. 2. The GPS loggers we used for our experiments were RoyalTek RBT-2300 models using the SiRF Star III chipset with WAAS enabled. These are relatively accurate and provide data with a standard deviation of about 4.1 meters according to our experiments.



(a) Microsoft shuttle (b) Paratransit van

Fig. 2. A Microsoft shuttle is shown in (a), and a King County paratransit van is shown in (b)

The Microsoft shuttles roam around Redmond, Washington, USA, continuously during the day, and the paratransit vehicles move around Seattle, Washington, USA when they are called for service pickups.

Microsoft Shuttles: Shuttles provide both fixed and on-demand service between Microsoft campus buildings during the day. The GPS loggers mounted on these shuttles record time-stamped latitude/longitude coordinates with an interval of one second. We collected the recorded data over two weeks. From each shuttle we retrieved an average of about 358,300 time stamped latitude/longitude pairs, which corresponds to about 99.5 hours of data from each of the 55 shuttles. Most of the shuttles automatically turned off logging when they were parked for the night.

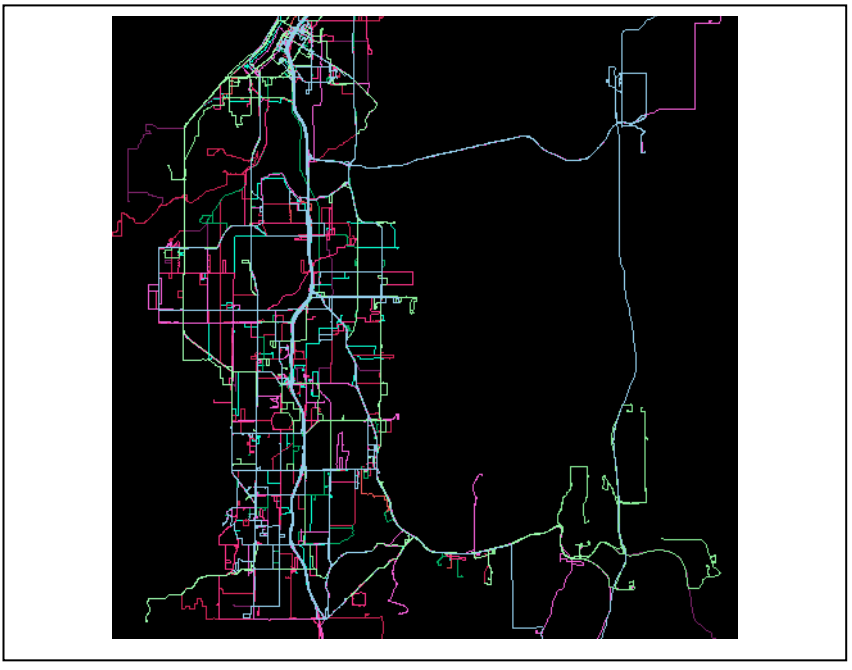


Fig. 3. This image shows trips segmented from our GPS data in different colors

King County Paratransit Vehicles: Paratransit vehicles provide on-demand services in Seattle and surrounding areas during the day and night. The GPS loggers mounted on these vehicles record time-stamped latitude/longitude coordinates with an interval of 5 seconds. We collected the recorded data over four weeks. From each vehicle we got an average of about 319,596 time stamped latitude/longitude pairs, which corresponds to about 444 hours of data from each of the 252 paratransit vehicles. The paratransit vehicles recorded continuously, which caused their logger memories to fill even when parked for the night. This is why we increased the sampling period to five seconds for these vehicles, as opposed to the shuttles' one second sampling period. Other than the sampling interval, the GPS loggers were identical for all the vehicles in our study. The two sets of vehicles have overlapping service areas, so we combined their data into one large data set.

As our first processing on this data, we split the traces into individual trips. Each trip consists of a sequence of GPS points which start when the vehicle starts moving and stops when the vehicle stops. We split the data into trips by looking at the gaps between timestamps to find when the vehicle was turned off. However, as mentioned previously, some loggers did not shut off automatically while the vehicles were parked, which provides us data while they were idle. To remove this useless data, we put a constraint on the minimum speed of the vehicle. Also sometimes we get very noisy latitude/longitude points from the loggers. To get rid of the noisy data, we also put a constraint on the maximum speed of the vehicles. Specifically, we split the data into trips whenever we find a gap of at least 10 seconds or the speed is slower than 5

miles per hour (mph) or faster than 90 mph. Retaining only the trip data means we can ignore data for parked vehicles and outliers, leading to faster downstream processing. Fig. 1(a) shows all the GPS trips for the Redmond and Seattle area in our database. We have depicted trips with different colors on a small area on the map in Fig. 3.

We use this data for detecting the locations of intersections. Having detected the intersections, we find the roads that connect them, and use these to refine the locations of the intersections. The next section describes how we detect intersections, followed by sections on refining their locations.

4 Intersection Detection

Our goal is to detect road intersections in GPS data. In a graph representation of the road network, intersections are nodes which are connected by road segments that serve as edges. An intersection is a location where more than two edges connect to each other. We describe in this section how it is possible to determine if an arbitrary location on the map is an intersection or non-intersection by looking at the local GPS traces around that point. After finding the intersections, we connect them based on information provided by the GPS traces. Then we refine the locations of the intersections. This section describes the first step of this process, which is detecting the intersections. Section 4.1 describes our detection process generally, and Section 4.2 gives details and results.

4.1 Intersection Detection Algorithm

Our intersection detector works by sliding a specialized, 2D circular window, called a local shape descriptor, over the GPS data to find places whose GPS data indicates an intersection. The shape descriptor describes the distribution of GPS points at each location on the map. The goal is to find a shape descriptor that (1) can perfectly discriminate between intersections and non-intersections and (2) can be represented as a feature vector which lets us apply machine learning to develop a classifier out of the positive and negative examples in our training data.

The basic idea behind the shape descriptor is illustrated in Fig. 4. Given a set of GPS edges (two temporally adjacent GPS points from the same vehicle), the shape descriptor captures the local distribution and direction of GPS edges in the circle around that location. Our shape description is a set of annular sections, each of which can be thought of as a histogram bin. For every given edge, we add a point to each bin that the edge passes through.

We map the bins of each shape descriptor to a vector and then learn a classifier out of all the feature vectors provided by the training examples. To map the shape descriptor to the feature vector, there are a few issues that we must consider. First, we normalize the raw counts in the bins so they sum to one. This helps neutralize the differences between heavily traveled and lightly traveled intersections.

The other issue is that the shape descriptors are sensitive to the headings of the roads converging at the intersection. In our training, we want to limit the apparent diversity of intersections, so we attempt to compute a canonical orientation for each intersection and then rotate the corresponding shape description so its canonical

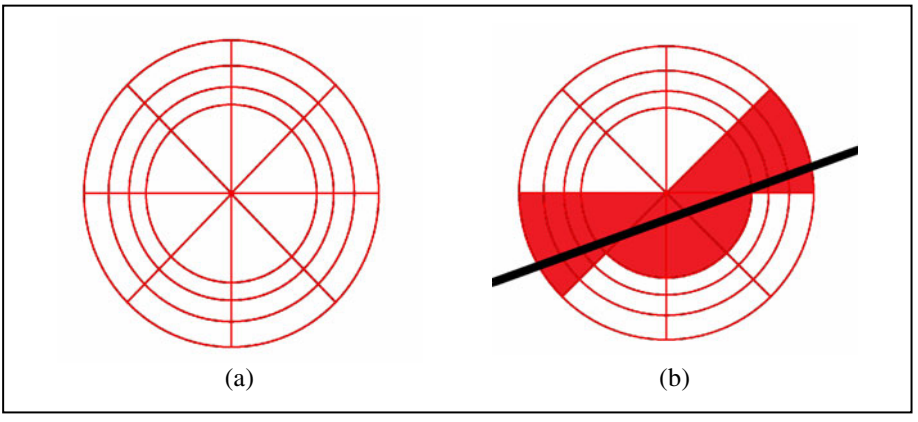


Fig. 4. The shape descriptor used for discriminating between intersections and non-intersections. (a) An example of shape descriptor with 32 bins. (b) For each edge passing from shape descriptor, a point is added to each of the bins the edge is passing through. The actual shape detector we used had each annulus split into 16 annular sections.

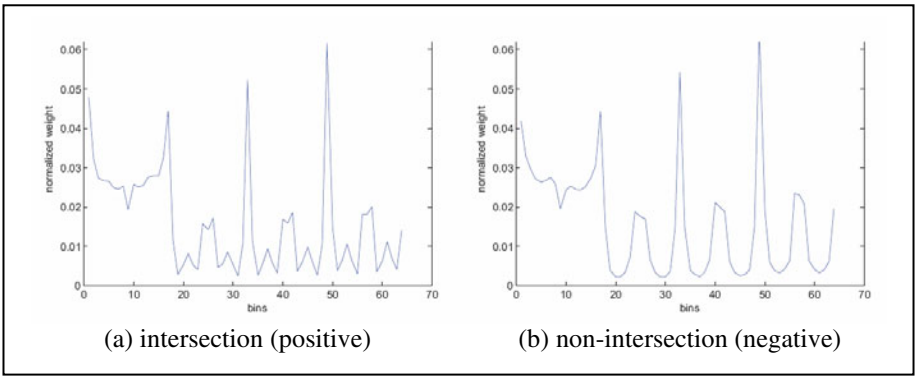


Fig. 5. Average of the 64 positive feature vectors is shown in (a) and average of 64 negative feature vectors is shown in (b). These are taken in annulus-major order, so bins 1-8 correspond to the inner annulus. As can be seen by comparing the plots, the positive features are more jagged. Because the negative samples are usually on straight roads, they have a peak every 180 degrees (every 8 bins), while the positive samples, which are 3- or 4-way intersections, have a peak usually every 90 degrees (every 4 bins).

orientation is zero. This helps make the intersection detector rotation invariant. We take the canonical orientation for each shape descriptor as the direction of the bin that has the maximum weight, and then we rotate the shape descriptor so this angle is zero. Then we insert the weights into the feature vector from those bins. In Fig. 5 we have shown average positive and negative examples of feature vectors.

We use the information provided by an available map to extract the ground truth location of intersections and the roads connecting the intersections to each other. Our

training data consists of positive examples taken from the locations of known intersections and negative examples taken along known roads that are at least 20 meters from any intersection. We look at all intersection types from our ground truth map, so our detector is not limited to any certain type of intersection.

To learn a classifier from the set of positive and negative feature vectors, we use the Adaboost algorithm, similar to that of Viola and Jones [12]. The Adaboost training algorithm learns a classifier as a subset of a group of weak classifiers that each can barely classify the data better than random. Adaboost adds the weak classifiers incrementally one at a time to the final classifier. At each iteration, the weight of each incorrectly classified example is increased, and the weak classifier that performs best on those is chosen to be added to the final classifier.

We consider the elements of the feature vector (*i.e.* normalized bins of the histogram) as potential weak classifiers $\{f_p\}$ for the Adaboost run. In each iteration t of the Adaboost training algorithm, one of the features $f_t \in \{f_p\}$ is chosen as the feature of the weak classifier $h_t(x)$ to be added to the final classifier. This weak classifier is of the form:

$$h_t(x) = \begin{cases} 1 & \text{if} \quad p_t f_t < p_t \theta_t \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\theta_t \in (-\infty, \infty)$ is the classification threshold of the classifier and $p_t = \pm 1$ is a parity for the inequality sign. The parity allows either $f_t(x) < \theta_t$ or $f_t(x) > \theta_t$ as a potential feature.

After all T iterations of the algorithm, we get the final classifier $H(x)$, which is of the form

$$H(x) = \begin{cases} 1 & \text{if} \quad \sum_{t=1}^{T} \alpha_t h_t(x) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where α_t is the selected weight for weak classifier $h_t(x)$, as chosen by the Adaboost algorithm. The final classifier consists of a sum of the chosen weak classifiers. We can replace ≥ 0 in Equation (2) with $\geq \lambda$ to adjust the bias of the classifier to make it more or less sensitive at the cost of more false positives or more false negatives.

After learning the classifier that can discriminate between the intersections and non-intersections, we test it by moving it over the GPS traces. (Note that the training locations are at ground truth locations of intersections and roads, while the test locations are taken at the actual GPS measurements.) For every location we check, we fill the shape descriptors bins, normalize, rotate to a canonical orientation, and pass the feature vector through our classifier. The classifier $H(x)$ returns a number in the range of (−1, 1) for each test location. The locations with a greater value will have a higher probability of being an intersection. For each of the tested locations, we look at the value of the other locations around it. If there was a location nearby with a higher value, we discard the current location. We do the same for all the locations and prune the ones which are not a local maximum. This helps eliminate tight clusters of intersections. We accept the remaining locations as intersections if their classification value $H(x)$ is higher than a threshold λ.

We believe this is the first application of machine learning and a shape descriptor to the problem of finding intersections from GPS data.

The following section describes the specifics of our implementation of this approach to detecting intersections along with performance results.

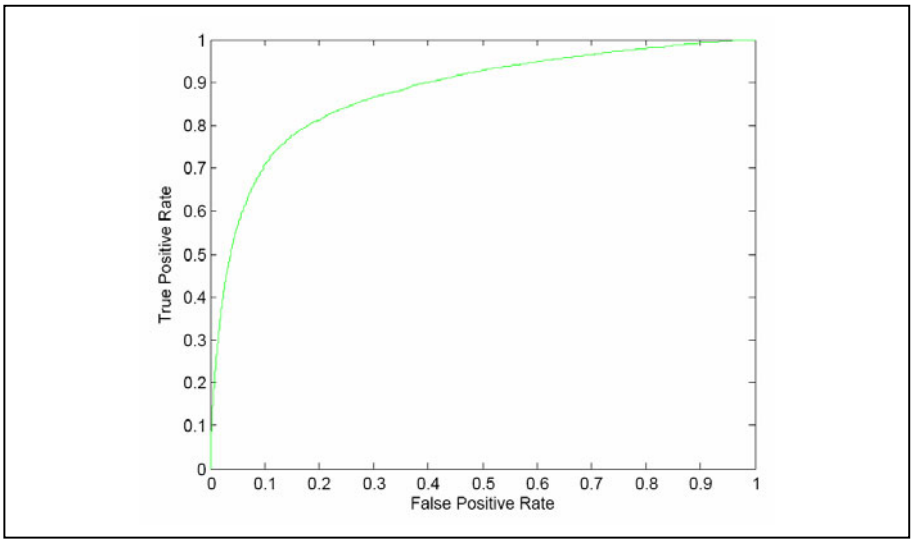


Fig. 6. This ROC curve shows the tradeoff between true positives and false positives for intersection detection with different values of the classifier threshold λ. The ideal operating point would be in the upper left corner with all true positives and no false negatives.

4.2 Intersection Detection Specifics and Performance

The shape descriptor has a few parameters that we need to tune, which are the number of circles, the radius of the smallest circle, the ratio between the radii of circles, and the number of angular slices. To discover the best set of parameters, we sweep through these parameters, training a classifier on half of the training data and measuring performance on the other half. The following parameters for the shape descriptor yielded the best performance: 4 circles, smallest radius of 20 meters, ratio between radii of circles of 1.3, and 16 angular slices.

Using the shape descriptor parameters just discovered, our next step is to learn a classifier using all the training data. The training data consists of positive examples which are shape descriptors centered at ground truth locations of known intersections. However, we do not have GPS data for all the area, because some intersections were never visited by our GPS loggers. Also, for some of the visited intersections, there is not data available for all roads connected to that intersection. We prune these before learning the classifier. The negative examples are taken from shape descriptors put on the ground along known roads at least 20 meters from any intersection.

We learn a classifier using 2000 iterations of the Adaboost algorithm. During the test, we look at all the GPS data. We sample points on the GPS traces, taking a point

as a sample if there is not a point sampled already within 5 meters. After filling the shape descriptor for a sampled point, we apply our classifier to measure the likelihood of the point being an intersection. The classifier assigns a value between −1 and 1, which we threshold with the value λ to determine whether or not the point is an intersection. The performance of the classifier is shown in Fig. 6 using an ROC curve that demonstrates performance for different values of the threshold λ. If a point is within 10 meters of a ground truth node, we declare that it should be labeled positive, otherwise it should be labeled negative. Section 6 describes how we refine the locations of these intersections.

We remove the points that have another point with a higher classifier output within 20 meters or whose classifier output value is less than $\lambda = 0.03$. This replaces multiple positive responses close to an intersection with a single detected intersection.

The next step in our overall process is to find roads that run between the intersections we found, which we describe in the next section.

5 Finding Roads

With intersections found, we next find roads that connect the intersections. While our main goal in this paper is to detect intersections, discovering the roads between intersections is a necessary prerequisite for our final step of refining the intersections' locations. This section describes how we filled in the roads between intersections and presents quantitative results comparing the lengths of our discovered roads to their true lengths.

5.1 Road Finding Algorithm

Each trip in our database represents a vehicle moving from one location to another. We use the trips to connect the intersections to each other. Each trip may pass through a number of intersections.

For each trip, we find the intersections that the trip passes through. These intersections are the ones that are near the path taken by the vehicle on that trip. We sort these intersections based on their path distance from the beginning of the trip, and we connect each intersection to its adjacent ones in the trip. While many trips from various vehicles may connect two adjacent intersections, we choose the one trip with the minimum distance between the two intersections as the best one.

This algorithm may cause some mistaken connections. Assume intersection A is connected to intersection C through intersection B. However, a trip might be noisy and may not pass close enough to B. This causes our algorithm to connect A to C, which is not the case in ground truth. To prune such mistakes, for every two connected nodes M and N, we find their shortest indirect path, *i.e.* the path that connects them through another intersection. If the shortest indirect path is almost the same length as the direct path, we remove the direct path. This is because, in this case, the direct path likely missed an intervening intersection slightly.

Another mistake we correct is some of the false positive intersections we found based on the shape descriptor method in Section 4.1. These false intersections will have only two roads connected to them, which violates our definition of an intersection. Thus, we prune all intersections that have two or fewer connected roads.

Fig. 7. After connecting the intersections using the road filling algorithm, there are some mistaken connections. In (a) we have shown the road network before pruning incorrect connections. In (b) the road network is depicted after removing incorrect connections.

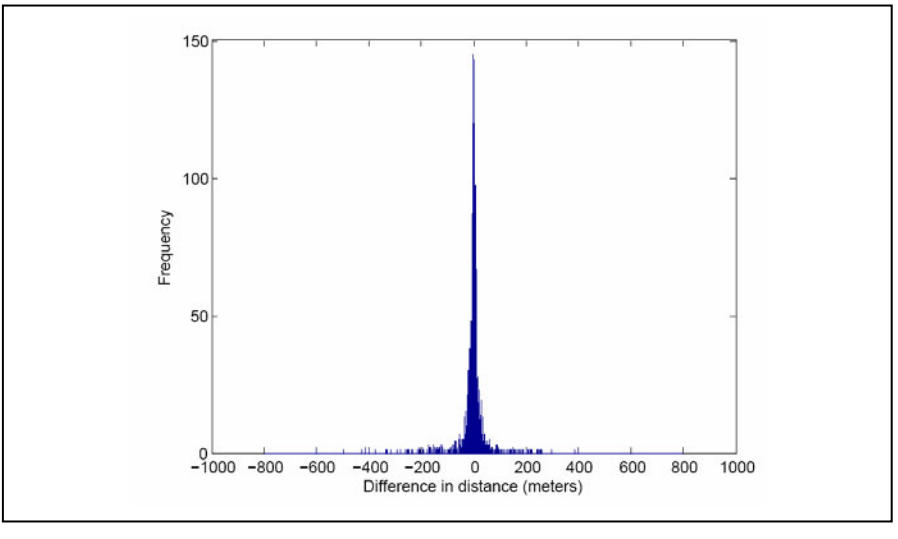


Fig. 8. This is the histogram of differences between path distance in the computed road network and ground truth. The errors are tightly clustered around zero.

5.2 Road Finding Specifics and Results

The result of road filling algorithm is presented in Fig. 7. If a trip passes closer than 30 meters to an intersection, it is assumed to be passing through that intersection. We remove a road between intersections A and node C with length l if there is another path between A and C with a length less than $\sqrt{2}l$. We remove the direct connection in this case because it means there are other intersections between A and C. We also remove an intersection if there are less than three roads connected to it. We see from Fig. 7 that the pruning technique effectively removes many mistaken roads. To measure the performance of road filling step, we compare the path distance between corresponding intersections in the computed road network to ground truth. We find the path distance between two intersections using the classic Dijkstra algorithm. In Fig. 8 we have shown the histogram of differences between path distances of

corresponding intersections in the computed road network and ground truth. The errors are tightly centered around zero.

These roads are valuable for refining the locations of the detected intersections. We discuss this process in the next section.

6 Refining Intersection Locations

In Section 4 we found the intersections and in Section 5 we connected the intersections with roads. The location of the intersections may be mistaken by a few meters, since we apply our intersection detector on a grid with 5 meter spacing. In this section we present an algorithm based on Iterative Closest Point (ICP) [13] to optimize the locations of the intersections.

6.1 Intersection Refinement Algorithm

The ICP algorithm is used to match two clouds of points. It iteratively matches the points in the model and the ones in the data, and it estimates the rigid transformation (translation and rotation) that moves the model to the data. In our specific case, the data is the raw GPS traces. The model is the detected intersection and the set of roads we found in Section 5 that connect to the detected intersection. Around each intersection we sample points on the GPS traces and project them on the roads connected to that intersection. Each sampled GPS point corresponds to its projection on the local road network around the intersection. Then we find a transformation matrix that maps the points on the model to the data points. We transform the intersection and the connected roads with this matrix. We keep iterating this procedure until convergence. The algorithm is as below:

1. Associate each point in the GPS traces (D) to its nearest neighbor in the road network (X) discovered in Section 5. D and X are $3 \times N$ matrices. Column i of D is a homogenous coordinate point in the data corresponding to column i of X, which is a homogenous point in the model. (A homogeneous coordinate has a 1 appended at the end, *e.g.* $(x, y, 1)^T$.)
2. Estimate the transformation matrix using the standard minimum squared error criterion:

$$T^* = \underset{T}{\operatorname{argmin}} |D - TX|_2 = DX^T(XX^T)^{-1}$$

3. Transform the model (*i.e.* intersection and connected roads) using the estimated transformation matrix: $X_{new} = T^* X_{previous}$.
4. If X is moved significantly, go back to step 1, otherwise terminate.

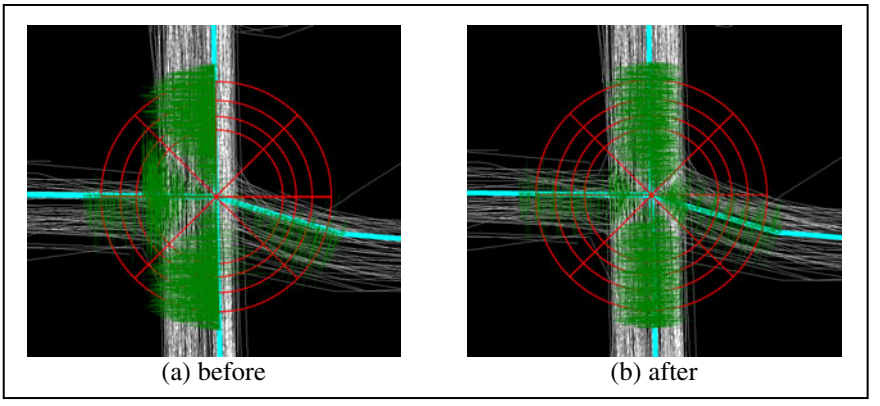


Fig. 9. The result of applying the ICP algorithm to optimize the node locations. In (a) the estimated node is a few metersoff from the ground truth. In (b) the location of the node is fixed after the ICP algorithm is applied.

In Fig. 9 we have shown the result of applying the ICP step to an intersection which is a few meters off from where it is supposed to be based on the GPS traces. The white lines represent the GPS traces, the red circle is the shape descriptor, the cyan edges show the road network around the intersection, and the green lines connect the corresponding points on the data and model.

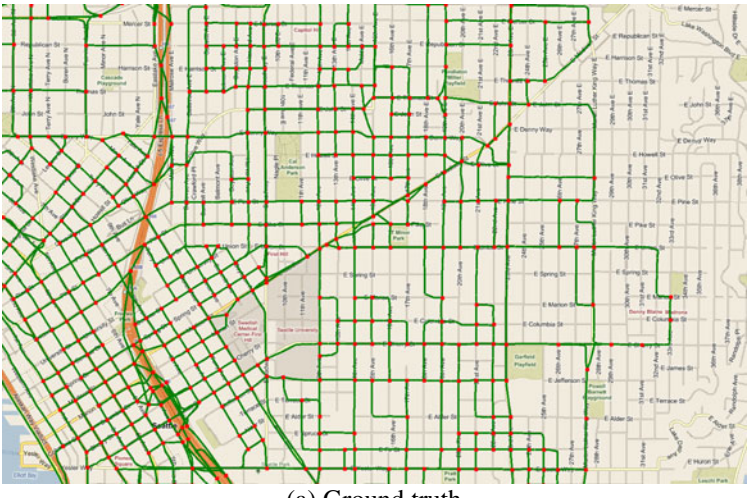
After we optimize the location of the intersections using the ICP algorithm, we next recompute the edges between the intersections. We perform the road filling algorithm described in Section 5 to recalculate the roads. We iteratively optimize the intersection locations using the ICP and recalculate the roads until convergence.

6.2 Intersection Refinement Results

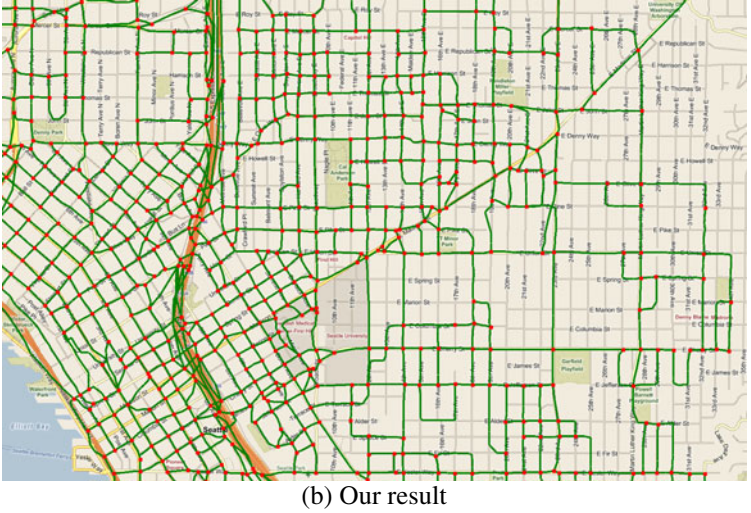
For each intersection, we find the correspondence between the GPS data around it and its connecting roads. We consider the local GPS points in an annulus with inner radius 25 meters and outer radius 40 meters, centered on the intersection. We do not consider the GPS points closer than 25 meters since they fall inside the intersection and make it difficult to find the correct corresponding connected road.

We applied the ICP algorithm to all our detected intersections that were within 20 meters of a ground truth intersection, which amounted to 76% of the ground truth intersections. After applying ICP, the average distance between the detected intersections and their ground truth locations improved from 7.2 meters to 4.6 meters, indicating that our refinement approach had a significant positive impact on the accuracy of the intersections' locations. Part of the remaining error could be due to inaccuracies in the underlying map.

We have compared the final results of our approach with the ground truth in Fig. 10.



(a) Ground truth



(b) Our result

Fig. 10. (a) Ground Truth image of the intersections and the roads connecting them. (b) The results computed by our algorithm. We remove from the ground truth the intersections and roads for which we do not have enough GPS data.

7 Conclusions

In this paper we have demonstrated a new pattern recognition method for finding road intersections based on GPS traces of everyday drivers. Our approach is an alternative to the costly and slow method of using dedicated drivers and vehicles for the purpose of map generation.

We find intersections from GPS data in three steps. First we find the intersections using a classifier learned over the shape descriptors. In the next step we connect the intersections with roads. In the last step we optimize the estimated road network to best fit the connected roads.

Future work should include an assessment of what types of intersections are most often found or missed with our approach, in particular highway merges where the exact intersection point is not as clear as a regular surface road intersection.

References

1. Haklay, M., Weber, P.: OpenStreetMap: User-Generated Street Maps. In: IEEE Pervasive Computing, pp. 12–18 (2008)
2. Tavakoli, M., Rosenfeld, A.: Building and Road Extraction from Aerial Photographs. IEEE Transactions on Systems, Man, and Cybernetics SMC-12(1), 84–91 (1982)
3. Hu, J., et al.: Road Network Extraction and Intersection Detection From Aerial Images by Tracking Road Footprints. IEEE Transactions on Geoscience and Remote Sensing 45(12), 4144–4157 (2007)
4. Barsia, A., Heipkeb, C.: Artificial Neural Networks for the Detection of Road Junctions in Aerial Images. In: The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Munich, Germany, pp. 113–118 (2003)
5. Rogers, S., Langley, P., Wilson, C.: Mining GPS Data to Augment Road Models. In: International Conference on Knowledge Discovery and Data Mining, pp. 104–113 (1999)
6. Guo, T., Iwamura, K., Koga, M.: Towards High Accuracy Road Maps Generation from Massive GPS Traces Data. In: IEEE International Geoscience and Remote Sensing Symposium, pp. 667–670 (2007)
7. Bruntrup, R., et al.: Incremental Map Generation with GPS Traces. IEEE Intelligent Transportation Systems, 574–579 (2005)
8. Worrall, S., Nebot, E.: Automated Process for Generating Digitised Maps through GPS Data Compression. In: Australasian Conference on Robotics and Automation, Brisbane, Australia (2007)
9. Schroedl, S., et al.: Mining GPS Traces for Map Refinement. Data Mining and Knowledge Discovery 9(1), 59–87 (2004)
10. Edelkamp, S., Schrödl, S.: Route Planning and Map Inference with Global Positioning Traces. In: Klein, R., Six, H.-W., Wegner, L. (eds.) Computer Science in Perspective. LNCS, vol. 2598, pp. 128–151. Springer, Heidelberg (2003)
11. Cao, L., Krumm, J.: From GPS Traces to a Routable Road Map. In: 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPAITLA GIS 2009), pp. 3–12. ACM, Seattle (2009)
12. Viola, P., Jones, M.: Rapid Object Detection using a Boosted Cascade of Simple Features. In: Computer Vision and Pattern Recognition (CVPR 2001), p I-511– I-518 (2001)
13. Besl, P.J., McCay, N.D.: A Method for Registration of 3-D Shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14(2), 239–256 (1992)